04

Para saber más: Mass Assignment Attack

Mass Assignment Attack o Ataque de asignación masiva, en español, ocurre cuando un usuario logra inicializar o reemplazar parámetros que no deben ser modificados en la aplicación. Al incluir parámetros adicionales en una solicitud, si dichos parámetros son válidos, un usuario malintencionado puede generar un efecto secundario no deseado en la aplicación.

El concepto de este ataque se refiere a cuando inyectas un conjunto de valores directamente en un objeto, de ahí la asignación masiva de nombres, que sin la debida validación puede causar serios problemas.

Tomemos un ejemplo práctico. Suponga que tiene el siguiente método, en una clase Controller, utilizado para registrar un usuario en la aplicación:

```
@PostMapping
@Transactional
public void registrar(@RequestBody @Valid Usuario usuario) {
    repository.save(usuario);
}
```

COPIA EL CÓDIGO

Y la entidad JPA que representa al usuario:

```
@Getter
@Setter
@NoArgsConstructor
@EqualsAndHashCode(of = "id")
@Entity(name = "Usuario")
@Table(name = "usuarios")
```

```
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private String email;
    private Boolean admin = false;

    //restante del código omitido…
}
```

COPIA EL CÓDIGO

Observe que el atributo `admin` de la clase `Usuario` se inicializa como `falso`, lo que indica que un usuario siempre debe estar registrado como administrador. Sin embargo, si se envía el siguiente JSON en la solicitud:

```
{
    “nombre” : “Rodrigo”,
    “email” : “rodrigo@email.com”,
    “admin” : true
}
```

COPIA EL CÓDIGO

El usuario se registrará con el atributo `admin` con valor `true`. Esto sucede porque el atributo `admin` enviado en el JSON existe en la clase que se está recibiendo en el Controller, considerándose un atributo válido y que se llenará en el objeto `Usuario` que será instanciado por Spring.

Entonces, ¿cómo prevenimos este problema?

Prevención

El uso del patrón DTO nos ayuda a evitar este problema, ya que al crear un DTO definimos solo los campos que se pueden recibir en la API, y en el ejemplo anterior el DTO no tendría el atributo `admin`.

Nuevamente, vemos una ventaja más de usar el patrón DTO para representar los datos que entran y salen de la API.